|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

**Отчёт по лабораторной работе №3 «Программирование JAVA сокетов» по дисциплине «Архитектура клиент-серверных приложений»**

**Студент группы** ИКБО-10-19 Полетаев Владимир Викторович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

**Руководитель практической работы** преподаватель Степанов П.В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись руководителя)

Работа представлена «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Допущен к работе «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Москва 2021

**Цель работы:**

Знакомство студентов с одной из базовых технологий реализации конечной точки для передачи и приёма данных по сети – сокетом.

**Теоретическое введение:**

Клиент-серверная архитектура — это самая известная структура приложений в Интернете. В этой архитектуре клиенты (например, персональные компьютеры, устройства IoT и т. Д.) вначале запрашивают ресурсы с сервера, а затем сервер отправляет соответствующие ответы на запросы клиентов. Чтобы это произошло, как на клиентах, так и на серверах должен быть реализован какой-то механизм, поддерживающий эту сетевую транзакцию. Этот механизм называется коммуникацией через сокеты.

Рассмотрим более подробно процесс подключения с использованием сокетов JAVA, как на стороне клиента, так и на стороне сервера.

*На стороне сервера*:

Обычно сервер работает на определенном компьютере, и он создает «серверный сокет» (связанный с портом) и приостанавливает работу. Сервер регистрирует свою службу под номером порта. Затем сервер ожидает эту службу. Класс ServerSocket используется на стороне сервера: он просто ожидает вызовов от клиента (ов).

*На стороне клиента*:

Клиенты знают имя хоста сервера и номер порта, с которым сервер работает. Клиент может установить соединение с сервером, запросив создание Socket, предназначенного для сервера для порта, на котором была зарегистрирована служба. Клиент подключается к сокету сервера; Затем создаются два сокета: «клиентский сокет» на стороне клиента и «сокет клиентской службы» на стороне сервера. Эти сокеты соединены друг с другом

**Постановка задачи:**

Необходимо создать клиент-серверное приложение на языке JAVA с использованием socket, для широковещательного общения пользователей. Приложение может быть, как консольным, так и оснащённым полноценным GUI. Клиентское приложение считывает данные из стандартного ввода и отсылает сообщение серверу (с помощью TCP/IP). Сервер, в свою очередь, накапливает сообщения и раз в 5 секунд осуществляет массовую рассылку всем клиентам. Если сообщений за указанный период не поступило, то рассылка не производится. Клиент, получивший сообщение, отображает на экране текст данного сообщения. Структуру и поведение данного клиент-серверного приложения, в том числе, например, в части регистрации конкретного клиента и формата широковещательного сообщения, студент определяет самостоятельно.

**Код программы:**

Создаём сервер, который будет находиться в ожидании соединения со стороны клиента. При получении сообщения, поток останавливается на 5 секунд, давая возможность другим потокам (один поток на клиента) внести новые сообщения в List. После чего идёт массовая рассылка сообщений всем клиентам:

package com.example.demo.Pr3;  
  
import java.io.\*;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.ArrayList;  
import java.util.LinkedList;  
import java.util.List;  
  
public class Server {  
 public static final int *PORT* = 8080;  
 public static LinkedList<ServerFunction> *serverList* = new LinkedList<>();  
 public static List<String> *messages* = new ArrayList<>();  
  
 public static void main(String[] args) throws IOException {  
 ServerSocket server = new ServerSocket(*PORT*);  
 System.*out*.println("Server have started");  
 try {  
 while (true) {  
 Socket socket = server.accept();  
 try {  
 *serverList*.add(new ServerFunction(socket));  
 } catch (IOException e) {  
 socket.close();  
 }  
 }  
 } finally {  
  
 server.close();  
 }  
 }  
}  
  
class ServerFunction extends Thread {  
 private Socket socket;  
 private BufferedReader in;  
 private BufferedWriter out;  
  
 public ServerFunction(Socket socket) throws IOException {  
 this.socket = socket;  
 in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));  
 start();  
 }  
  
 @Override  
 public void run() {  
 String word;  
 try {  
 try {  
 while (true) {  
 Server.*messages*.clear();  
 word = in.readLine();  
 if (!word.equals("")) {  
 Server.*messages*.add(word);  
 System.*out*.println("New message: " + word);  
 Thread.*sleep*(5000);  
 // Отсылка каждому пользователю обратно  
 for (ServerFunction vr : Server.*serverList*) {  
 vr.send(Server.*messages*);  
 }  
 }  
 }  
 } catch (NullPointerException ignored) {  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 } catch (IOException e) {  
 this.down();  
 }  
 }  
  
 private void send(List<String> msg) {  
 try {  
 if (!msg.isEmpty()) {  
 out.write(msg + "\n");  
 out.flush();  
 }  
 } catch (IOException ignored) {  
 }  
 }  
  
 private void down() {  
 try {  
 if (!socket.isClosed()) {  
 socket.close();  
 in.close();  
 out.close();  
 for (ServerFunction vr : Server.*serverList*) {  
 if (vr.equals(this)) vr.interrupt();  
 Server.*serverList*.remove(this);  
 }  
 }  
 } catch (IOException ignored) {  
 }  
 }  
}

Далее нужен класс-клиент, который при установке соединения отправляет, либо же выводит сообщения:

package com.example.demo.Pr3;  
  
import java.io.\*;  
import java.net.\*;  
  
class ClientFunction {  
 // сокет - это одна конечная точка двустороннего канала связи между двумя программами,  
 // работающими на разных компьютерах в сети.  
 // Обычно сервер работает на определенном компьютере в сети и имеет сокет,  
 // привязанный к определенному номеру порта.  
  
  
 private Socket socket;  
 private BufferedReader in; // из сокета  
 private BufferedWriter out; // в сокет  
 private BufferedReader inputUser; // консоль  
  
 public ClientFunction(String addr, int port) {  
 try {  
 this.socket = new Socket(addr, port);  
 } catch (IOException e) {  
 System.*err*.println("Socket failed");  
 }  
 try {  
 inputUser = new BufferedReader(new InputStreamReader(System.*in*));  
 in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));  
 new readWord().start();  
 new writeWord().start();  
 } catch (IOException e) {  
 ClientFunction.this.down();  
 }  
 }  
  
 private void down() {  
 try {  
 if (!socket.isClosed()) {  
 socket.close();  
 in.close();  
 out.close();  
 }  
 } catch (IOException ignored) {  
 }  
 }  
  
 private class readWord extends Thread {  
 @Override  
 public void run() {  
 String str;  
 try {  
 while (true) {  
 str = in.readLine();  
 if (str.equals("stop")) {  
 ClientFunction.this.down();  
 break;  
 }  
 System.*out*.println(str);  
 }  
 } catch (IOException e) {  
 ClientFunction.this.down();  
 }  
 }  
 }  
  
 public class writeWord extends Thread {  
 @Override  
 public void run() {  
 while (true) {  
 String userWord;  
 try {  
 userWord = inputUser.readLine();  
 if (userWord.equals("stop")) {  
 out.write("stop" + "\n");  
 ClientFunction.this.down();  
 break;  
 } else {  
 out.write(userWord + "\n");  
 }  
 out.flush();  
 } catch (IOException e) {  
 ClientFunction.this.down();  
 }  
 }  
 }  
 }  
}  
  
public class Client {  
 public static String *ipAddr* = "localhost";  
 public static int *port* = 8080;  
  
 public static void main(String[] args) {  
 new ClientFunction(*ipAddr*, *port*);  
 }  
}  
  
class Client2 {  
 public static String *ipAddr* = "localhost";  
 public static int *port* = 8080;  
  
 public static void main(String[] args) {  
 new ClientFunction(*ipAddr*, *port*);  
 }  
}

**Результат работы программы**

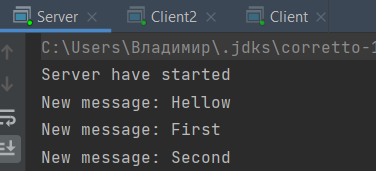


Рис. 1 Скриншот результатов работы программы

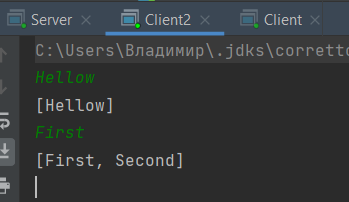


Рис. 2 Скриншот результатов работы программы

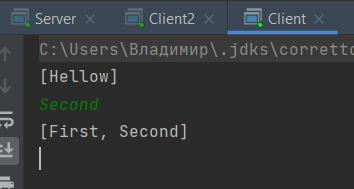


Рис. 3 Скриншот результатов работы программы

**Выводы:**

В работе была изучена и применена на практике реализация конечной точки для передачи и приёма данных по сети – сокетом.